GRAPHICS PROGRAMMING, LAB 1

GETTING STARTED IN PROCESSING

NAME: ______________________________

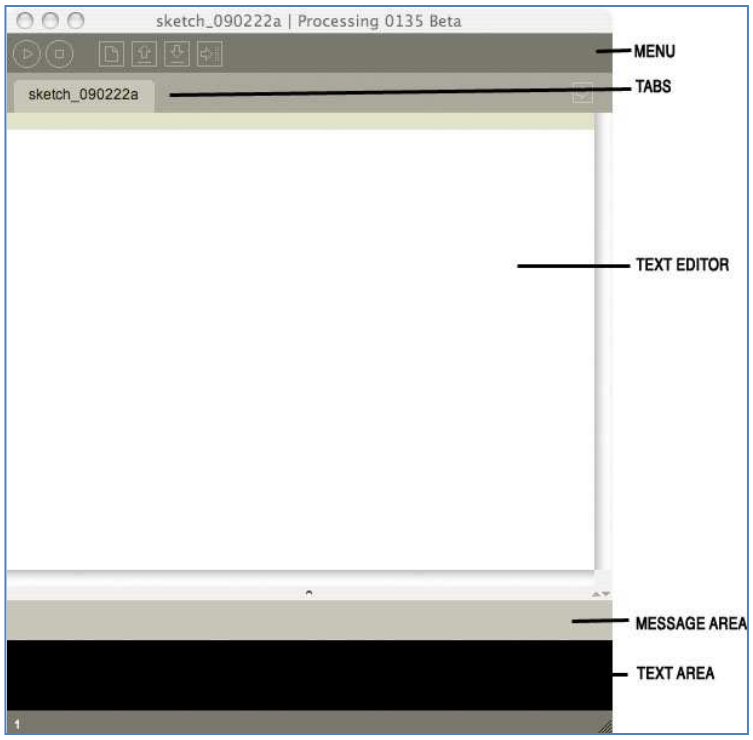
1. Overview

Processing is a “sketch” programming tool designed for use by non-technical people (e.g., artists, designers, musicians). For technical people, it is a handy tool for prototyping applications in Java. You can do lots of things with Processing. This lab will focus on drawing graphics.

1.1. Start up Processing:

- Double-click the Processing icon, which probably looks something like this ----------------------------->

1.2. The Processing window looks like this:



Note: The annotations on the right in the image to the left that point out the areas of the window.

1.3. Menu buttons (at top of window):

- **run** - compiles the code, opens a display window and runs the program.
- **stop** - terminates a running program.
- **new** - creates a new “sketch” in the current window.
- **open** - open files from your “sketchbook”, or another on your computer.
- **save** - saves the current sketch with its current name and location.
- **export** - exports the current sketch as a Java “applet”.

1.4. Notes on Menu buttons:

- **run** — Hold down the shift key to Present instead of run
- **new** — To create a new sketch in its own (new) window, use File – New
- **open** — Note that opening a sketch from the toolbar will replace the sketch in the current window. To open a sketch in a new window, use File - Open.
- **save**— If you want to give the sketch a different name, use File - Save As.

2. Your first program:

2.1. Writing the code to draw a line.

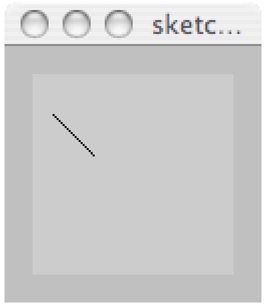
- In the text window, type the following:

```
line( 10, 20, 30, 40 );
```

- Note the punctuation—parenthesis, commas and semi-colon.
- Note that the "function" line is written in *lower case*. Processing is case-sensitive, so watch the case with each new function introduced.

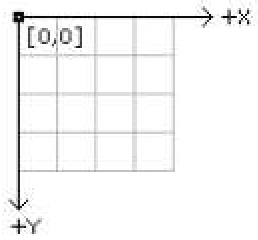
2.2. Running your program

- After you type the above in the text window, click on the **run** button.
- Processing should open a display window that looks like this:



2.3. Understanding what you have done.

- The **line**() function takes four "arguments". These are the endpoints of a line. Imagine that the display window is a piece of graph paper with two axes: x and y. The x axis runs horizontally along the display window, starting with 0 on the left and increasing as you move to the right. The y axis runs vertically down the display window, starting with 0 on the top and increasing as you move down.



- So, you have just drawn a line from (x1, y1) = (10, 20) to (x2, y2) = (30, 40).

2.4. Modifying your program

- Try changing the values of the arguments to the **line()** function to different (x, y) values. Click the **run** button to see the effect of the changes you made.
- Try adding a second **line()** function (put it on another line in the text editor, below the first **line()** function).

3. Adding color

3.1. The Processing function for drawing in color is called stroke(). It takes one argument, a 6-digit *hexademical* number specifying the amount of *red*, *green* and *blue* in the color that should be used for drawing.

3.2. Try adding a stroke() function call above your call(s) to line() in the text editor.

- Example:

```
stroke( #ff0000 );
line( 10,20,30,40 );
```

- *Does this remind you of setting colors in HTML/CSS?*

3.3. Check out the command Tools - Color Selector.

- You'll find help for picking cool colors!

3.4. Practice:

- Try adding multiple stroke() calls to your sketch, one before each line() call, each one with a different color. This way, each line you draw will be a different color.
- Try drawing a green square using one call to stroke() and four calls to line().
- Try drawing a square with each side a different color. You will need four calls to stroke() and four calls to line().
- *Hint: plan out your code ahead of time by drawing a square on paper and figuring out what the coordinates of each of the four corners should be.*

4. Learning new functions

4.1. Look up the syntax of each of the following functions on the Processing on-line reference page: http://www.processing.org/reference/index.html

4.2. Try putting each of the following commands in your sketch and see how they work.

- rect()
- ellipse()
- fill()
- triangle()
- strokeWeight()

4.3. Use several commands together to draw a simple shape like a house or a snowman.

5. Animation Programs

5.1. In the previous sections (above), you learned how to write programs that were static. That is, nothing ever changed in the application. In this section, you will learn how to write program that contains animation (what is drawn will change over time).

5.2. As above, remember to be mindful of upper and lower case, as well as punctuation.

5.3. Basic outline of a program:

-
  ```
  void setup() {

  }

  void draw() {

  }
  ```

5.4. Every processing animation program **must have** two functions.

- **setup()** -> The setup() function contains commands that you want computer to execute once when the sketch first starts.
- **draw()** -> After setup() is called, then draw() we be called repeatedly until the sketch is stopped. By default draw() is called 60 times a second.

5.5. Create a new sketch and enter the following code in the text window. Before you execute the program, take a guess as to what you think the program does.

-
  ```
  void setup() {
          background( #ff0000 );
  }

  void draw() {
          rect(mouseX, mouseY, 10, 10);
  }
  ```

6. Making your program interactive

6.1. In the previous sections (above), you learned how to write programs that had output. This means the programs display stuff. You used the coordinates of the mouse to change where a rectangle was drawn. In this section, you will learn how to write programs that take other types of input, which means that you can have the program, respond to things the user/viewer does like typing on the keyboard.

6.2. As above, remember to be mindful of upper and lower case, as well as punctuation.

6.3. Event Listener – keyPressed()

- Create a new sketch and enter the following code in your text window.

```
void setup() {
        background( #ff0000 );
}

void draw() {

}

void keyPressed() {
        background( #0000ff );
}
```

- Click on the **run** button, and then when Processing opens the display window, click anywhere in the display window and then click on any key. The color of the display window should change from red to blue.
- The code in the **setup()** function runs as soon as the sketch starts.
- The function **draw()** does nothing, but it is still necessary to keep the program open and listening for keyboard input. More information on the draw() function will be made available later.
- The code in the **keyPressed()** function runs as soon as the user presses a key. Note that you (the user) have to click in the display window to give it *focus*, so that the sketch will recognize (i.e., be "listening") when a key is pressed.

6.4. Making choices

- The above program demonstrates input from the keyboard, when *any* key is pressed.
- Now try entering and running the code contained in the box below, which responds differently when different keys are pressed.
- *Note:* The program will respond to the keys uppercase keys R, G, B, and W.
- Don't forget to click in the display window, before pressing any keys.
- As you create and run the program, observe the code that you are entering. You will see the words **if** and **else**. These are called *control structures*, and they control the flow of the code. If the user presses the R key, one thing happens (what is it?); otherwise, if the user presses the G key, something else happens (what is it?); and so on. The if...else statement is called a *conditional* control structure, because it specifies what the program should do under conditions specified by the programmer.

```
void setup() {
        background( #000000 );
}

void draw() {
        rect(mouseX, mouseY, 10, 10);
}

void keyPressed() {
        if ( key == 'R' ) {
                background( #ff0000 );
        }
        else if ( key =='G' ) {
                background( #00ff00 );
        }
        else if ( key == 'B' ) {
                background( #0000ff );
        }
        else if ( key == 'W' ) {
                background( #ffffff );
        }
}
```

- Try changing the code by modifying what happens when the user presses R. Note that whatever code you add/change, all functions have to be contained within curly brackets ({ and }). Currently, only the statement background(#ff0000); is in between the curly brackets. If you add more lines of code, keep them between the same curly brackets.
- Try modifying the code so that it responds to both uppercase and lowercase input (e.g., R and r). To do that change the if statement too: (key == 'R' || key =='r')
- Now try changing the code by adding another condition of your own, when another key is pressed (other than R, G, B, or W).

7. Programming challenge

7.1. Combine what you have learned so far to make a program that draws different shapes depending on what letters the user types.

- Example 1: Have your program draw a line if the user types L and a circle if the user types C.
- Example 2: Have your program draw more complicated graphics, like drawing a house if the user types H or a bridge if the user types B.
- Have fun!

8. On-line references

8.1. Processing

- Processing web site: http://www.processing.org/
- Getting started tutorial: http://www.processing.org/learning/gettingstarted/
- Drawing tutorial: http://www.processing.org/learning/drawing/
- Color tutorial: http://www.processing.org/learning/color
- Reference: http://www.processing.org/reference/index.html

8.2. Processing for mobile devices: http://mobile.processing.org/

9. Exporting an Applet/Application

9.1. Try clicking on the export button or selecting File - Export.

- Depending on the version of Processing that you are working in this will create a standalone application or a webpage that you can use to transfer your program to other systems. Give it a try.